4.3 Families of Requirements

Text for Chapter 4.3

[Bre01b] Bredereke, J. *A tool for generating specifications from a family of formal requirements*. In Kim, M., Chin, B., Kang, S., and Lee, D., editors, "Formal Techniques for Networked and Distributed Systems", pp. 319–334. Kluwer Academic Publishers (Aug. 2001).

A tool for families of CSP-OZ specifications.

Additional Background for Chapter 4.3

[Bre02] Bredereke, J. *Maintaining telephone switching software requirements*. IEEE Commun. Mag. **40**(11), 104–109 (Nov. 2002).

Telephone switching system structure problems and solutions.

[Zav01] Zave, P. *Requirements for evolving systems: A telecommunications perspective*. In "5th IEEE Int'l Symposium on Requirements Engineering", pp. 2–9. IEEE Computer Society Press (2001).

Feature-oriented descriptions and "feature engineering" in telephone switching.

[Mil98] Miller, S. P. *Specifying the mode logic of a flight guidance system in CoRE and SCR*. In "Second Workshop on Formal Methods in Software Practice", Clearwater Beach, Florida, USA (4–5 Mar. 1998).

Application of the CoRE approach to an auto-pilot.

[Bre00b] Bredereke, J. *genFamMem 2.0 Manual – a Specification Generator and Type Checker for Families of Formal Requirements*. University of Bremen (Oct. 2000). URL http://www.tzi.de/~brederek/genFamMem/.

Definition of CSP-OZ language extension and manual for the genFamMem tool.

[Bre00a] Bredereke, J. *Families of formal requirements in telephone switching*. In Calder, M. and Magill, E., editors, "Feature Interactions in Telecommunications and Software Systems VI", pp. 257–273, Amsterdam (May 2000). IOS Press.

Families of CSP-OZ specifications.

[Bre00d] Bredereke, J. *Specifying features in requirements using CSP-OZ*. In Gilmore, S. and Ryan, M., editors, "Proc. of Workshop on Language Constructs for Describing Features", pp. 87–88, Glasgow, Scotland (15–16 May 2000). ESPRIT Working Group 23531 – Feature Integration in Requirements Engineering.

Families of CSP-OZ specifications.

[Bre00c] Bredereke, J. *Hierarchische Familien formaler Anforderungen*. In Grabowski, J. and Heymer, S., editors, "Formale Beschreibungstechniken für verteilte Systeme – 10. GI/ITG-Fachgespräch", pp. 31–40, Lübeck, Germany (June 2000). Shaker Verlag, Aachen, Germany.

Families of CSP-OZ specifications, ordered hierarchically.

[Bre01a] Bredereke, J. *Ein Werkzeug zum Generieren von Spezifikationen aus einer Familie formaler Anforderungen*. In Fischer, S. and Jung, H. W., editors, "Formale Beschreibungstechniken – 11. GI/ITG-Fachgespräch", Bruchsal, Germany (June 2001). URL http://www.i-u.de/fbt2001/.

A tool for families of CSP-OZ specifications.

[Bre99] Bredereke, J. *Modular, changeable requirements for telephone switching in CSP-OZ*. Tech. Rep. IBS-99-1, University of Oldenburg, Oldenburg, Germany (Oct. 1999).

Case study with families of CSP-OZ specifications.

[Bre98] Bredereke, J. *Requirements specification and design of a simplified telephone network by Functional Documentation*. CRL Report 367, McMaster University, Hamilton, Ontario, Canada (Dec. 1998).

Case study with families of Parnas tables.

[Kat93] Katz, S. *A superimposition control construct for distributed systems*. ACM Trans. Prog. Lang. Syst. **15**(2), 337–356 (Apr. 1993).

Seminal paper on superimposition.

[BrSc02] Bredereke, J. and Schlingloff, B.-H. *An automated, flexible testing environment for UMTS*. In Schieferdecker, I., König, H., and Wolisz, A., editors, "Testing of Communicating Systems XIV – Application to Internet Technologies and Services", pp. 79–94. Kluwer Academic Publishers (Mar. 2002).

Families of CSP test specifications.

Overview of Chapter 4.3

- feature-oriented description
- the CoRE method
- families of CSP-OZ specifications
- families of CSP test specifications

Focus on Requirements

- motivation:
  - ○ all feature interaction problems already (implicitly) present in requirements
  - ○ many "formal methods" support single product only
    - ▷ how to integrate family support into method?

Feature-Oriented Description in Telephone Switching

- base description plus separate feature descriptions
- attraction: behavioural "modularity"
  - ○ easy change of system behaviour
  - ○ make *any* change by just adding a new feature description
  - ○ never change existing descriptions
- emphasizes individual features
  - ○ makes them explicit
- de-emphasizes feature interactions
  - ○ makes them implicit in the feature composition operator

- not all feature interactions are bad
  - ○ feature-oriented description relies on the good ones
- example: busy treatments
  - ○ B_1 and B_2 both enabled, B_2 higher priority
  - ○ B_1 not applied, despite its stand-alone description
  - ○ behavioural "modularity": add new busy treatments without changing existing ones
- most feature-oriented descriptions still informal
  - ○ behavioural "modularity" and formality do not combine easily
    - ▷ behavioural "modularity": don't answer some questions now
    - ▷ formality: answer all questions now
  - ○ proposed composition operators / approaches often do not scale

- IP telephony:
  - ○ highly complex new services
  - ○ services still viewed as stand-alone
  - ○ undesired feature interactions will haunt us soon

Feature-Oriented Descriptions and Common Abstractions

- modules need common abstractions/assumptions
  - module: now in the sense of this lecture
  - common abstraction/assumption: true for *all* family members
- rapid innovation, legacy systems, too many players: hard to limit the domain
- *without domain limits: no common abstractions*

Performing Incremental Specification Formally

- standard means:
  stepwise refinement
- step:
  1. extend behaviour *or* 2. impose constraints
  - example 1.: add another potential event to a state
  - example 2.: specify the order of two events
- interesting properties preserved by step
  - example 1.: all old events remain possible
    - ▷ no deadlock in this state
  - example 2.: no harmful event added
    - ▷ all safety properties preserved

Non-Monotonous Changes

- telephone switching:
  new features change the behaviour
  - of base system, or
  - of other features
- example: call forwarding
  - stops to connect to dialled number
    - ▷ restricts base system behaviour

    and
  - starts connecting to forwarded-to number
    - ▷ extends base system behaviour

Formal Support for Feature Specification

- considerable research effort
  on feature composition operators
- FIREworks project
  (Feature Interactions in Requirements Engineering)
  - various feature operators proposed and investigated
- "feature-oriented programming"
- based on the superimposition idea by Katz
- analytical complexity:
  too big for tools for real systems

Superimposition

- by Katz [Kat93]
- approach:
  - base system
  - textual increments
  - composition operator
- problem:
  - increments have defined interface,
    base system has not
  - increment can invalidate arbitrary assumptions about base system

The CoRE Method

- based on four-variable model and SCR
- groups the variables into classes
- developed during the early 1990's
- no explicit family support, but maybe a good base for it
- no formal syntax and semantics
- no tool support

Families of CSP-OZ Specifications

key ideas:

- *maintain all variants together*
  - generate specific member automatically as necessary
- *document information needed for changes*
  - dependence of requirements
  - what is the core of a feature

Constraint-Oriented Specification

- features closely interrelated
  - most refer to mode of connection
  - user interface: few, shared lexical events
    - ▷ system cannot be sliced by controlled events
- incrementally impose partial, self-contained constraints
- composition by logical conjunction

The Formalism CSP-OZ

- CSP-OZ demo: one very simple telephone
- CSP-OZ class inheritance for incremental constraints

Case Study on Telephone Switching Requirements

- black box specification of telephone switching
- attempt to incorporate new concepts
- details: see [Bre99]
  papers: see [Bre01b, Bre01a, Bre00c, Bre00a, Bre00a]

Grouping Classes into Features

the chapters of the requirements document:

1. Introduction
2. `feature` UserSpace
3. `feature` BasicConnection
4. `feature` VoiceChannel
5. `familymember` SpecificationA
6. `feature` ScreeningBase
7. `feature` BlackListOfDevices
8. `familymember` SpecificationB
9. `feature` BlackListOfUsers
10. `feature` FollowHumanConnectionForwarding
11. `familymember` SpecificationC
12. `feature` TransferUserRoleToAnotherHuman
13. `familymember` SpecificationD

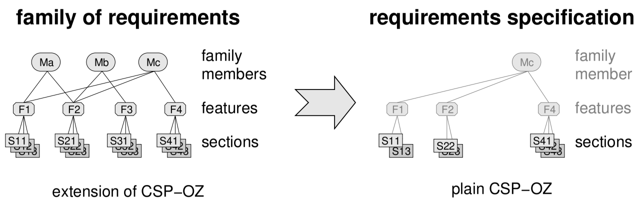
⋮ ⋮

Indices / Bibliography

The Feature Construct

- feature UserSpace
- feature BasicConnection
- familymember SpecificationB

Generating Family Members From a Family Document



Result of Family Member Generation

1. Introduction
2. `feature` UserSpace
3. `feature` BasicConnection
4. `feature` VoiceChannel
5. `feature` ScreeningBase
6. `feature` BlackListOfDevices
7. `familymember` SpecificationB

Indices / Bibliography

- family member composition chapter:
  part replaced

Controlled Non-Monotonous Changes

- feature ScreeningBase
- feature BlackListOfUsers
- feature FollowHumanConnectionForwarding
- familymember SpecificationC

Avoiding Feature Interactions

- introduced three notions explicitly
  - “telephone device”
  - “human”
  - “user role”
- consequences:
  - black list above:
    screens user roles, not devices
  - another black list feature:
    screens devices, not user roles
  - also two kinds of call forwarding
- no feature interaction screening–forwarding anymore

Detecting Feature Interactions by Type Checks

- *type rules*: part of the family extension of CSP-OZ
- syntactic rules → *syntactic errors*:
  - ◦ “remove” an “essential” class
  - ◦ feature of needed class not included
  - ◦ feature of “removed” class not included
  - ◦ another class still needs “removed” class
- heuristic syntactic rules → *syntactic warnings*:
  - ◦ class is marked both essential and changeable
  - ◦ class is “removed” twice

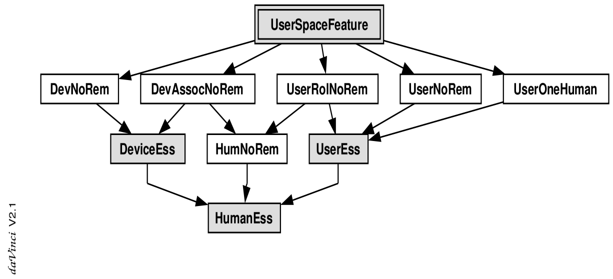
Feature Interactions Detected in Case Study

- no interactions between TCS and CF
  - ◦ no type errors detectable
- but other problems problems present:
  - ◦ both screening features “remove” the same section
  - ◦ type rules: warning!
  - ◦ manual inspection: contradiction
- resolution: another feature

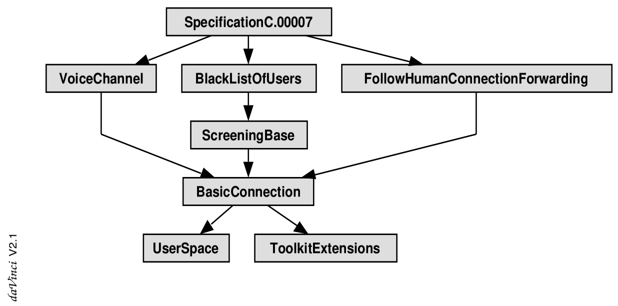
Documenting Dependences

- uses-relation for requirements:
  - ◦ use of previous definition
  - ◦ reliance on previous constraint
- documented by:
  - ◦ Z’s section “parents” construct
  - ◦ class inheritance (mapped to Z sections)
- if no relationship: identifiers out of scope

Sections of Feature UserSpace



Hierarchy of Features of SpecificationC



Hierarchical Requirements Specification

- a feature can build on other features
- in contrast to the Intelligent Network
- possible to have feature providing a common base

The Tool genFamMem 2.0

- extracts specifications in plain CSP-OZ from a family document,
- detects feature interactions by
  - ◦ additional type checks for families
  - ◦ heuristic warnings
- helps avoiding feature interactions by generating documentation on the structure of the family.
- available freely

Further Tools

- cspozTC
  - ◦ type checker for CSP-OZ
- daVinci
  - ◦ visualizes uses hierarchy graphs

Semantics of CSP-OZ Extension

- formal definition of language extension in [Bre00b]
  - understand details: need to know Object-Z and CSP

What Is Still To Do?

- more experience – extend case study further
- apply to other formalisms than CSP-OZ
  - necessary:
    constraint-oriented specification style
    and incremental refinement
  - already supported: CSP_Z and plain Z
- investigate relationship:
  families of requirements – families of programs

Families of CSP Test Specifications

- testing of embedded systems with RT-Tester tool
- RLC layer in UMTS protocol stack
- project with Bosch/Siemens Salzgitter
- requirements specification in CSP
- see [BrSc02]

- light-weight application of previous ideas
  - no consistency checks
  - no documentation generation
  - simple preprocessor for CSP plus method

Flexible Maintenance of Test Specification

- late changes to requirements
- variants of test suites:
  (a) adjust test coverage
    ▷ selected signal parameters
    ▷ stimuli: random → increased probabilities → deterministic
  (b) component / integration tests
    ▷ different protocol layers
    ▷ parallel instances of same layer
  (c) active / passive tests

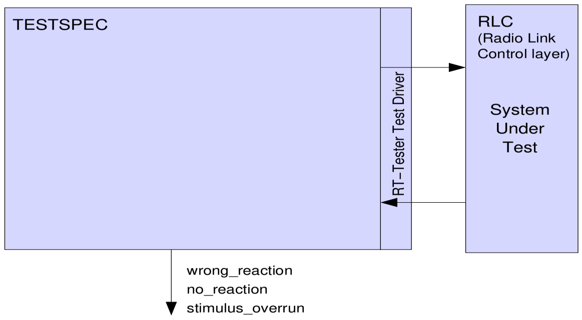
  ⇒ a family of test suites

Rules for Modularizing Requirements

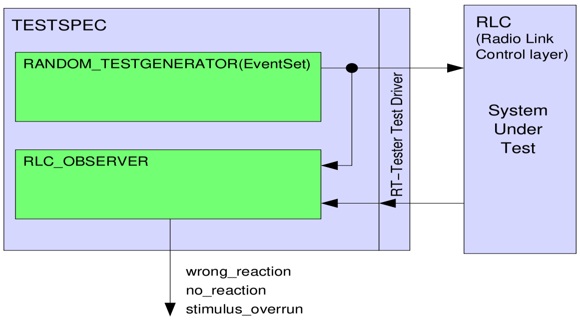
- separate: signature / behaviour of module
- identify requirements that will change together, put into one module

  specifically, separate:
  - tester specific issues / application
  - timer handling / application
  - protocol layers
  - stimulus generation / test observation

Separate: Test Stimulus Generation / Test Observation



Cont.: Separate: Test Stimulus Generation / Test Observation



Summary of Lecture

- safety-critical systems
  - quality does matter
- professional engineering
  - "blueprint before build"
    ▷ Chapter 2: what information in computer system documentation?
- embedded software systems
  - "ugly", strict interface constraints
    ▷ Chapter 1: rigorous description of requirements
  - interface changes all the time
    ▷ Chapter 3: decomposition into modules
    ▷ Chapter 4: families of systems

5. Appendix

References

[Bre98] Bredereke, J. *Requirements specification and design of a simplified telephone network by Functional Documentation*. CRL Report 367, McMaster University, Hamilton, Ontario, Canada (Dec. 1998).

[Bre99] Bredereke, J. *Modular, changeable requirements for telephone switching in CSP-OZ*. Tech. Rep. IBS-99-1, University of Oldenburg, Oldenburg, Germany (Oct. 1999).

[Bre00a] Bredereke, J. *Families of formal requirements in telephone switching*. In Calder, M. and Magill, E., editors, "Feature Interactions in Telecommunications and Software Systems VI", pp. 257–273, Amsterdam (May 2000). IOS Press.

[Bre00b] Bredereke, J. *genFamMem 2.0 Manual – a Specification Generator and Type Checker for Families of Formal Requirements*. University of Bremen (Oct. 2000). URL http://www.tzi.de/~brederek/genFamMem/.

[Bre00c] Bredereke, J. *Hierarchische Familien formaler Anforderungen*. In Grabowski, J. and Heymer, S., editors, "Formale Beschreibungstechniken für verteilte Systeme – 10. GI/ITG-Fachgespräch", pp. 31–40, Lübeck, Germany (June 2000). Shaker Verlag, Aachen, Germany.

[Bre00d] Bredereke, J. *Specifying features in requirements using CSP-OZ*. In Gilmore, S. and Ryan, M., editors, "Proc. of Workshop on Language Constructs for Describing Features", pp. 87–88, Glasgow, Scotland (15–16 May 2000). ESPRIT Working Group 23531 – Feature Integration in Requirements Engineering.

[Bre01a] Bredereke, J. *Ein Werkzeug zum Generieren von Spezifikationen aus einer Familie formaler Anforderungen*. In Fischer, S. and Jung, H. W., editors, "Formale Beschreibungstechniken – 11. GI/ITG-Fachgespräch", Bruchsal, Germany (June 2001). URL http://www.i-u.de/fbt2001/.

[Bre01b] Bredereke, J. *A tool for generating specifications from a family of formal requirements*. In Kim, M., Chin, B., Kang, S., and Lee, D., editors, "Formal Techniques for Networked and Distributed Systems", pp. 319–334. Kluwer Academic Publishers (Aug. 2001).

[Bre02] Bredereke, J. *Maintaining telephone switching software requirements*. IEEE Commun. Mag. **40**(11), 104–109 (Nov. 2002).

[BrSc02] Bredereke, J. and Schlingloff, B.-H. *An automated, flexible testing environment for UMTS*. In Schieferdecker, I., König, H., and Wolisz, A., editors, "Testing of Communicating Systems XIV – Application to Internet Technologies and Services", pp. 79–94. Kluwer Academic Publishers (Mar. 2002).

[Cou85] Courtois, P.-J. *On time an space decomposition of complex structures*. Commun. ACM **28**(6), 590–603 (June 1985).

[HBPP81] Heninger Britton, K., Parker, R. A., and Parnas, D. L. *A procedure for designing abstract interfaces for device interface modules*. In "Proc. of the 5th Int'l. Conf. on Software Engineering – ICSE 5", pp. 195–204 (Mar. 1981).

[HoWe01] Hoffman, D. M. and Weiss, D. M., editors. *Software Fundamentals – Collected Papers by David L. Parnas*. Addison-Wesley (Mar. 2001).

[JaKh99] Janicki, R. and Khedri, R. *On a formal semantics of tabular expressions*. CRL Report 379, McMaster University, Hamilton, Ontario, Canada (Sept. 1999).

[Kat93] Katz, S. *A superimposition control construct for distributed systems*. ACM Trans. Prog. Lang. Syst. **15**(2), 337–356 (Apr. 1993).

[Lam88] Lamb, D. A. *Software Engineering: Planning for Change*. Prentice-Hall (1988).

[LaRö01] Lankenau, A. and Röfer, T. *The Bremen Autonomous Wheelchair – a versatile and safe mobility assistant*. IEEE Robotics and Automation Magazine, "Reinventing the Wheelchair" **7**(1), 29–37 (Mar. 2001).

[Mil98] Miller, S. P. *Specifying the mode logic of a flight guidance system in CoRE and SCR*. In "Second Workshop on Formal Methods in Software Practice", Clearwater Beach, Florida, USA (4–5 Mar. 1998).

[PaCl86] Parnas, D. L. and Clements, P. C. *A rational design process: how and why to fake it*. IEEE Trans. Softw. Eng. **12**(2), 251–257 (Feb. 1986).

[PaMa95] Parnas, D. L. and Madey, J. *Functional documents for computer systems*. Sci. Comput. Programming **25**(1), 41–61 (Oct. 1995).

[Par72] Parnas, D. L. *On the criteria to be used in decomposing systems into modules*. Commun. ACM **15**(12), 1053–1058 (1972).

[Par74] Parnas, D. *On a 'buzzword': Hierarchical structure*. In "IFIP Congress 74", pp. 336–339. North-Holland (1974). Reprinted in [HoWe01].

[Par76] Parnas, D. L. *On the design and development of program families*. IEEE Trans. Softw. Eng. **2**(1), 1–9 (Mar. 1976).

[Par77] Parnas, D. L. *Use of abstract interfaces in the development of software for embedded computer systems*. NRL Report 8047, Naval Research Lab., Washington DC, USA (3 June 1977). Reprinted in Infotech State of the Art Report, Structured System Development, Infotech International, 1979.

[Par79] Parnas, D. L. *Designing software for ease of extension and contraction*. IEEE Trans. Softw. Eng. **SE-5**(2), 128–138 (Mar. 1979).

[PaWe85] Parnas, D. L. and Weiss, D. M. *Active design reviews: Principles and practices*. In "Proc. of the 8th Int'l Conf. on Software Engineering – ICSE 8", London (Aug. 1985).

[PCW85] Parnas, D. L., Clements, P. C., and Weiss, D. M. *The modular structure of complex systems*. IEEE Trans. Softw. Eng. **11**(3), 259–266 (Mar. 1985).

[Pet00] Peters, D. K. *Deriving Real-Time Monitors from System Requirements Documentation*. PhD thesis, McMaster Univ., Hamilton, Canada (Jan. 2000).

[vSPM93] van Schouwen, A. J., Parnas, D. L., and Madey, J. *Documentation of requirements for computer systems*. In "IEEE Int'l. Symposium on Requirements Engineering – RE'93", pp. 198–207, San Diego, Calif., USA (4–6 Jan. 1993). IEEE Comp. Soc. Press.

[WeLa99] Weiss, D. M. and Lai, C. T. R. *Software Product Line Engineering – a Family-Based Software Development Process*. Addison Wesley Longman (1999).

[Zav01] Zave, P. *Requirements for evolving systems: A telecommunications perspective*. In "5th IEEE Int'l Symposium on Requirements Engineering", pp. 2–9. IEEE Computer Society Press (2001).